JACOBS UNIVERSITY

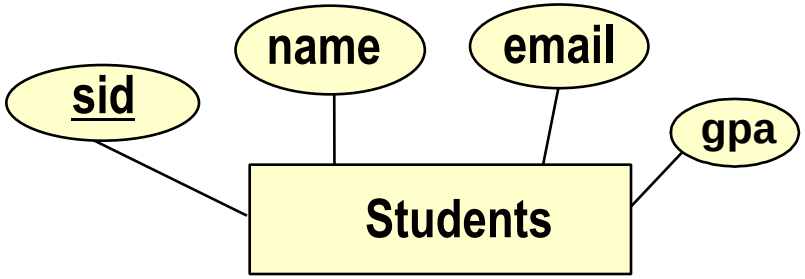
QUERYING DATABASES

Objectives

- After this unit you will be able:
  - to read and write simple SQL queries
  - including joins

Logical DB Design: ER to Relational

- ER attribute → relational attribute:
  - Decide about type now
- Entity set → table:
  - One entity set = 1 table
  - Map attributes
  - Pick primary key
- Relationship set → table:
  - One relship set = 1 table
  - Map attributes
  - Relship legs → foreign keys




```
CREATE TABLE Students
(  sid INTEGER, name CHAR(11),
   email CHAR(11), gpa DECIMAL(2)
   PRIMARY KEY (sid)
)
```

```
CREATE TABLE Enrolled
( sid INTEGER, cid INTEGER, grade DECIMAL(2),
  PRIMARY KEY (sid,cid),
  FOREIGN KEY (sid) REFERENCES Students
 )
```

Primary Keys

- Primary key = identifier for a tuple
  - Yes, there are secondary keys ☺
- Possibly many candidate keys, DB designer chooses
  - ..or creates an artificial one (best)
- PK gets specific support for fast search

  → index (later)

```
CREATE TABLE Student
  ( sid INTEGER,
    name  CHAR(20),
    email CHAR(20),
    gpa DECIMAL(2),
    PRIMARY KEY  (sid)
)
```

```
Students:
sid name     email     gpa
-----------------------
42  Neo      n.eo      1.7
43  Trinity  t.rinity  2.3
44  Cypher   c.ypher   3.0
```

Foreign Keys in SQL

- Foreign Key = attribute referencing key of some tuple
  - Same or different relation
- Ex: “Only students listed in Students can enroll for courses”

Just to show compound p‘keys

```
CREATE TABLE EnrolledIn
 ( sid INTEGER,  cid INTEGER,  grade DECIMAL(2),
   PRIMARY KEY  (sid,cid),
   FOREIGN KEY (sid) REFERENCES Students )
```

Students:

sid	name	email	gpa
42	Neo	n.eo	1.7
43	Trinity	t.rinity	2.3
44	Cypher	c.ypher	3.0

EnrolledIn:

sid	cid	grade
42	142	1.3
43	142	2.0
44	142	2.0
42	143	1.0
44	143	2.0
43	144	1.7

Querying Relational Databases

- major strength of relational model: simple, powerful querying of data
  - Query reads tables, returns a table
  - Small set of generic operations, work on any table structure
- Query describes structure of result ("what"),
  not algorithm computing result ("how")
  - Simpler than detailed programs; data independence; optimizability; …
- Queries can be written more intuitively,
  DBMS responsible for efficient evaluation
  - key: precise (mathematical) semantics for relational queries
  - Allows optimizer to extensively re-order operations

Joining Tables

- Cross Product = several tables in query, return all tuple combinations
  - FROM clause
- Join = cross product, but conditions between tuples restrict result set
- Most common: equi-join: *R1.attr1 = R2.attr2*
- Ex: „Students enrolled in some course“

```
SELECT  S.name
FROM    Student S, EnrolledIn E
WHERE  S.sid=E.sid
```

Conceptual Query Evaluation

- Semantics of an SQL query defined through following conceptual evaluation strategy:
  - cross-product of *relation-list*
  - Discard tuples failing qualification
  - Delete attributes not in *target-list*
  - If *DISTINCT*, eliminate duplicate rows

SELECT	[DISTINCT] *target-list*
FROM	*relation-list*
WHERE	*qualification*

- probably least efficient way to compute a query!
  - optimizer will find more efficient strategies

Sample Evaluation

Students:

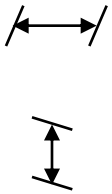
sid	name	email	gpa
42	Neo	n.eo	1.7
43	Trinity	t.rinity	2.3
44	Cypher	c.ypher	3.0

EnrolledIn:

sid	cid	grade
42	142	1.3
43	142	2.0
44	142	2.0
42	143	1.0
44	143	2.0
43	144	1.7

Courses:

cid	cname	credits	instId
142	AlgDS	5	242
143	DBWA	5	243
144	CGVis	5	244



```
SELECT  S.name, C.cname
FROM    Students S, EnrolledIn E, Courses C
WHERE   S.sid=E.sid AND E.cid=C.cid AND E.grade<2.0
```

sid	name	email	gpa	sid	cid	grade	cid	cname	credits	instId
42	Neo	n.eo	1.7	42	142	1.3	142	AlgDS	5	242
43	Trinity	t.rinity	2.3	43	142	2.0	142	AlgDS	5	242
44	Cypher	c.ypher	3.0	44	142	2.0	142	AlgDS	5	242
42	Neo	n.eo	1.7	42	143	1.0	143	DBWA	5	243
44	Cypher	c.ypher	3.0	44	143	2.0	143	DBWA	5	243
43	Trinity	t.rinity	2.3	43	144	1.7	144	CGVis	5	244

Sample Evaluation

Students:

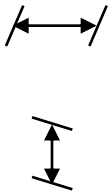
sid	name	email	gpa
42	Neo	n.eo	1.7
43	Trinity	t.rinity	2.3
44	Cypher	c.ypher	3.0

EnrolledIn:

sid	cid	grade
42	142	1.3
43	142	2.0
44	142	2.0
42	143	1.0
44	143	2.0
43	144	1.7

Courses:

cid	cname	credits	instId
142	AlgDS	5	242
143	DBWA	5	243
144	CGVis	5	244



```
SELECT  S.name, C.cname
FROM    Students S, EnrolledIn E, Courses C
WHERE   S.sid=E.sid AND E.cid=C.cid AND E.grade<2.0
```

name	cname
Neo	AlgDS
Neo	DBWA
Trinity	CGVis

Aggregation

- = deriving summary data
- Ex: „How many students pass the GenCS course?“

```
SELECT  count( S.sid )
FROM    Students S, EnrolledIn E, Courses C
WHERE  E.cname = ‘GenCS’ AND E.cid=C.cid AND E.sid=S.sid
```

- Several aggregation operations available
  - max, min, avg, sum, count
  - any, all

Changing Database Contents

- insert a single new tuple:

```
INSERT INTO Students( sid, name, email, gpa )
VALUES ( 51, 'Morpheus', 'm.orpheus', 1.0 )
```

- delete all tuples satisfying some condition:

```
DELETE FROM Students S
WHERE S.name = 'Cypher'
```

- change all tuples satisfying some condition:

```
UPDATE Students S SET gpa = 1.0
WHERE S.name = 'Neo'
```

Summary

- Relational model + SQL dominant in the market
  - Flexible + scalable + information integration + …
  - Simple, easy-to-grasp paradigm: *tables* → *flexible language* → *table*
- SQL is ISO standard →uniform interface
  - for DBMS implementers: clear guidance, leaving room for clever implementations
  - for application developers: no vendor lock-in
- BTW: Our group is preparing
  ISO 9075 SQL Part 15: Multi-Dimensional Arrays
  - Based on concepts of our rasdaman system, www.rasdaman.org